Contents

Fault Injection

5 minute read page test

This task shows you how to inject faults to test the resiliency of your application.

Before you begin

- Set up Istio by following the instructions in the Installation guide.
- Deploy the Bookinfo sample application including the default destination rules.

- Review the fault injection discussion in the Traffic Management concepts doc.
- Apply application version routing by either performing the request routing task or by running the following commands:

```
$ kubectl apply -f @samples/bookinfo/networking
/virtual-service-all-v1.yaml@
$ kubectl apply -f @samples/bookinfo/networking
/virtual-service-reviews-test-v2.yaml@
```

- With the above configuration, this is how requests flow:
  - `productpage` → `reviews:v2` → `ratings` (only for user `jason`)
  - `productpage` → `reviews:v1` (for everyone else)

Injecting an HTTP

delay fault

To test the Bookinfo application microservices for resiliency, inject a 7s delay between the `reviews:v2` and `ratings` microservices for user `jason`. This test will uncover a bug that was intentionally introduced into the Bookinfo app.

Note that the `reviews:v2` service has a 10s hard-coded connection timeout for calls to the `ratings` service. Even with the 7s delay that you introduced, you still expect the end-to-end flow to continue without any errors.

1. Create a fault injection rule to delay traffic coming from the test user `jason`.

   ```
   $ kubectl apply -f @samples/bookinfo/networking
   /virtual-service-ratings-test-delay.yaml@
   ```

2. Confirm the rule was created:

```
$ kubectl get virtualservice ratings -o yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
...
spec:
  hosts:
  - ratings
  http:
  - fault:
      delay:
        fixedDelay: 7s
        percentage:
          value: 100
    match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: ratings
        subset: v1
  - route:
    - destination:
        host: ratings
        subset: v1
```

Allow several seconds for the new rule

to propagate to all pods.

Testing the delay configuration

1. Open the Bookinfo web application in your browser.
2. On the `/productpage` web page, log in as user `jason`.

   You expect the Bookinfo home page to load without errors in approximately 7 seconds. However, there is a problem: the Reviews section displays an error message:

   ```
   Sorry, product reviews are currently unavailabl
   e for this book.
   ```

3. View the web page response times:
    1. Open the *Developer Tools* menu in you web browser.
    2. Open the Network tab
    3. Reload the `/productpage` web page. You will see that the page actually loads in about 6 seconds.

Understanding what happened

You've found a bug. There are hard-coded timeouts in the microservices that have caused the `reviews` service to fail.

As expected, the 7s delay you introduced doesn't affect the `reviews` service because

the timeout between the `reviews` and `ratings` service is hard-coded at 10s. However, there is also a hard-coded timeout between the `productpage` and the `reviews` service, coded as 3s + 1 retry for 6s total. As a result, the `productpage` call to `reviews` times out prematurely and throws an error after 6s.

Bugs like this can occur in typical enterprise applications where different teams develop different microservices independently. Istio's fault injection rules help you identify such anomalies without impacting end users.

> Notice that the fault injection test is restricted to when the logged in user is `jason`. If you login as any other user, you will not experience

any delays.

Fixing the bug

You would normally fix the problem by:

1. Either increasing the `productpage` to `reviews` service timeout or decreasing the `reviews` to `ratings` timeout
2. Stopping and restarting the fixed microservice
3. Confirming that the `/productpage` web page returns its response without any errors.

However, you already have a fix running in v3 of the `reviews` service. The `reviews:v3`

service reduces the `reviews` to `ratings` timeout from 10s to 2.5s so that it is compatible with (less than) the timeout of the downstream `productpage` requests.

If you migrate all traffic to `reviews:v3` as described in the traffic shifting task, you can then try to change the delay rule to any amount less than 2.5s, for example 2s, and confirm that the end-to-end flow continues without any errors.

Injecting an HTTP abort fault

Another way to test microservice resiliency is to introduce an HTTP abort fault. In this task, you will introduce an HTTP abort to

the `ratings` microservices for the test user `jason`.

In this case, you expect the page to load immediately and display the `Ratings service is currently unavailable` message.

1. Create a fault injection rule to send an HTTP abort for user `jason`:

    ```
    $ kubectl apply -f @samples/bookinfo/networking/virtual-service-ratings-test-abort.yaml@
    ```

2. Confirm the rule was created:

```
$ kubectl get virtualservice ratings -o yaml
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
...
spec:
  hosts:
  - ratings
  http:
  - fault:
      abort:
        httpStatus: 500
        percentage:
          value: 100
    match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: ratings
        subset: v1
  - route:
    - destination:
        host: ratings
        subset: v1
```

Testing the abort configuration

1. Open the `Bookinfo` web application in your browser.
2. On the `/productpage`, log in as user `jason`.

   If the rule propagated successfully to all pods, the page loads immediately and the `Ratings service is currently unavailable` message appears.
3. If you log out from user `jason` or open the Bookinfo application in an anonymous window (or in another browser), you will see that `/productpage` still calls `reviews:v1` (which does not call `ratings` at all) for everybody but `jason`. Therefore you will not see any error message.

Cleanup

1. Remove the application routing rules:

   ```
   $ kubectl delete -f @samples/bookinfo/networkin
   g/virtual-service-all-v1.yaml@
   ```

2. If you are not planning to explore any follow-on tasks, refer to the Bookinfo cleanup instructions to shutdown the application.